APC Injection

i) APC → Asynchronous Procedural call

- Asynchronous → not immediate, not blocking, happens later
- Procedural call → A request to execute a specific block of instructions [aka function/routine] in a defined execution context

• **Defined Execution context** → "when the worker pauses; run this task using everything they already have"

i.e. in tech terms

A procedure runs only inside a ~~prepared~~ ~~workspace~~ ready environment which includes

— A specific thread *

— CPU state → current possitions and registers

— Memory → variables and stack it can use already belonging to the process

— Privi ledge → user mode or kernel mode [i what it's allowed to do] *

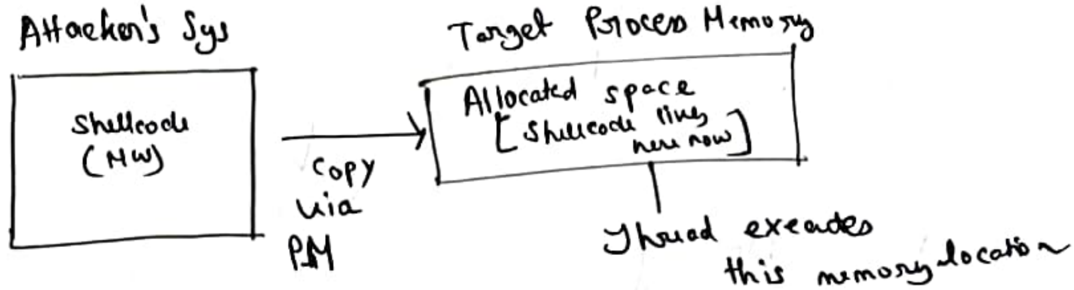
— Scheduling ~~rules~~ → rules → when and how long it can run *

> An employee at their ~~desk~~ desk, during work hours, using company tools, performs a task. She does not build a new office, issue new ID cards for every task.

• **Procedure** → code / instructions [Thread executes multiple procedures]

• **Thread** → ~~Pause~~ later...

APC → execution of predefined set of steps, at a later safe time



PM → Process Memory

[APC Injection in visuals representation]

Technical terms

Thread
- has a unique ID given by OS

A single thread can execute many procedures but not simultaneously

What thread contains
- Instruction pointer [where it is executing]
- CPU registers
- It's own stack

What thread shares
- Process memory
- Files
- Resources

As of Now we won't go in How it shares and How it contains

Same kitchen, Multiple chefs, shared ingredients

- Same kitchen → Process
- Multiple chefs → Multiple threads
- shared ingredients → Everything happening at the ~~same~~ same defined execution context

Thread switch task based on scheduling
→ Person with one name can do many tasks likewise with threads

How thread actually runs?

1. • User open chrome
2. OS loads chrome into memory [Ram] → Process created
   - OS kernel creates a process object
   - Allocates a new virtual address space
   - OS maps chrome's executable code, required libraries into that new virtual space
     - some part may stay on disk until needed (lazy loading)
   - (nutshell) OS loads a program, it maps its code into RAM so the CPU can execute it
3. → OS creates the main thread
4. → Thread starts executing instructions (Procedure)
5. → when chrome waits (I/O, user input) thread pauses
6. → OS switches CPU to another thread
7. → Later, chrome's thread resumes

Switching happens 1000 of times per second

Thread at instruction x
↓
calls procedure
↓
CPU executes procedure
↓
returns to instruction x+1

4 → Thread enters a procedure executes its instructions returns enters another procedure (repeats)

Process
↓
Thread — (executes procedures sequentially or via scheduling)
↓
Procedure — (Block of instructions)
↓
Instructions

for my convenience ← ignore it